

RECEIVED
CENTRAL FAX CENTER

MAR 03 2006

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P.O. Box 272400
Fort Collins, Colorado 80527-2400

PATENT APPLICATION

ATTORNEY DOCKET NO. 200308328-1IN THE
UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s): Cormac Flanagan et al

Confirmation No.: 7829

Application No.: 10/005923

Examiner: John J. Romano

Filing Date: Dec 04, 2001

Group Art Unit: 2182

Title: Method And Apparatus For Automatically Inferring Annotations

Mail Stop Appeal Brief-Patents
Commissioner For Patents
PO Box 1450
Alexandria, VA 22313-1450TRANSMITTAL OF APPEAL BRIEFTransmitted herewith is the Appeal Brief in this application with respect to the Notice of Appeal filed on Nov 8 2005.

The fee for filing this Appeal Brief is (37 CFR 1.17(c)) \$500.00.

(complete (a) or (b) as applicable)

The proceedings herein are for a patent application and the provisions of 37 CFR 1.138(a) apply.

☒ (a) Applicant petitions for an extension of time under 37 CFR 1.138 (fees: 37 CFR 1.17(a)-(d)) for the total number of months checked below:☒ 1st Month
\$120☐ 2nd Month
\$460☐ 3rd Month
\$1020☐ 4th Month
\$1580☐ The extension fee has already been filed in this application.☐ (b) Applicant believes that no extension of time is required. However, this conditional petition is being made to provide for the possibility that applicant has inadvertently overlooked the need for a petition and fee for extension of time.Please charge to Deposit Account 08-2025 the sum of \$ 620. At any time during the pendency of this application, please charge any fees required or credit any over payment to Deposit Account 08-2025 pursuant to 37 CFR 1.25. Additionally please charge any fees to Deposit Account 08-2025 under 37 CFR 1.18 through 1.21 inclusive, and any other sections in Title 37 of the Code of Federal Regulations that may regulate fees. A duplicate copy of this sheet is enclosed.☐ I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to:
Commissioner for Patents, Alexandria, VA 22313-1450
Date of Deposit:

OR

☒ I hereby certify that this paper is being transmitted to the Patent and Trademark Office facsimile number (571)273-8300.

Date of facsimile: March 3 2006

Typed Name: Carrie McKelvey

Signature: 

Respectfully submitted,

Cormac Flanagan et al.

By: 

Phillip S. Lyren

Attorney/Agent for Applicant(s)

Reg No.: 40,709

Date: March 3 2006

Telephone: 281 514 8236

Rev 10/06 (April 07)

03/07/2006 EFLORES 00000005 10005923

02 FC:1251 120.00 DA

CENTRAL FAX CENTER

MAR 03 2006

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants: Cormac A. Flanagan et al.

Examiner: John J. Romano

Serial No.: 10/005,923

Group Art Unit: 2192

Filed: December 4, 2001

Docket No.: 200308328-1

Title: Method and Apparatus for Automatically Inferring Annotations**APPEAL BRIEF UNDER 37 C.F.R. § 41.37**

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

This Appeal Brief is filed in response to the Final Office Action mailed August 10, 2005 and in response to the Notice of Panel Decision from Pre-Appeal Brief Review mailed January 5, 2006.

AUTHORIZATION TO DEBIT ACCOUNT

It is believed that no extensions of time or fees are required, beyond those that may otherwise be provided for in documents accompanying this paper. However, in the event that additional extensions of time are necessary to allow consideration of this paper, such extensions are hereby petitioned under 37 C.F.R. § 1.136(a), and any fees required (including fees for net addition of claims) are hereby authorized to be charged to Hewlett-Packard Development Company's deposit account no. 08-2025.

03/07/2006 EFLORES 00000005 082025 10005923

01 FC:1402 500.00 DA

03/07/2006 EFLORES 00000005 082025 10005923

02 FC:1851 120.00 DA

Application No. 10/005,923
Appcal Brief

I. REAL PARTY IN INTEREST

The real party-in-interest is the assignee, Hewlett-Packard Company, a Delaware corporation, having its principal place of business in Palo Alto, California.

II. RELATED APPEALS AND INTERFERENCES

There are no known related appeals or interferences known to appellant, the appellant's legal representative, or assignee that will directly affect or be directly affected by or have a bearing on the Appeal Board's decision in the pending appeal.

III. STATUS OF CLAIMS

Claims 1 – 51 stand finally rejected. The rejection of claims 1 – 51 is appealed.

IV. STATUS OF AMENDMENTS

All claims pending in this appeal are original. No amendments were made.

V. SUMMARY OF CLAIMED SUBJECT MATTER

The following provides a concise explanation of the subject matter defined in each of the claims involved in the appeal, referring to the specification by page and line number and to the drawings by reference characters, as required by 37 C.F.R.

§ 41.37(c)(1)(v). Each element of the claims is identified by a corresponding reference to the specification and drawings where applicable. Note that the citation to passages in the specification and drawings for each claim element does not imply that the limitations from the specification and drawings should be read into the corresponding claim element or that these are the sole sources in the specification supporting the claim features.

Claim 1

A method of annotating a computer program, comprising:

- a) applying a program checking tool to the computer program to produce one or more warnings (FIG. 4: #118, 402, 404; p. 16, lines 1-9);
- b) mapping at least one of said warnings into at least one annotation modification (FIG. 4: #410; p. 16, lines 9-14);

Application No. 10/005,923
Appeal Brief

c) modifying the computer program in accordance with said at least one annotation modification so that the number of annotations in the computer program changes, thereby producing a modified computer program (FIG. 4: #412; p. 16, lines 14-17);

d) repeating each of steps a), b) and c) until no warnings produced in step a) are suitable for mapping into an annotation modification (p. 15, lines 8-17; p. 16, lines 23-31); and

e) providing a user with the modified computer program in which is found at least one annotation (p. 16, lines 26-29).

Claim 3

The method of claim 2 wherein, prior to said mapping, said warnings about potential misapplications of primitive operations in the computer program are identified, and said modifying comprises inserting into the computer program at least one annotation that is produced by mapping at least one of said warnings about potential misapplications of primitive operations into an annotation modification (p. 6, lines 21-26; p. 12, lines 20-29).

Claim 6

The method of claim 5 wherein said warnings about inconsistencies between the computer program and one or more of the annotations are identified, and said modifying comprises removing from the computer program one of said heuristically derived annotations identified by said at least one annotation modification (p. 6, line 28 – p. 7, line 3; p. 15, lines 8-17).

Claim 22

A computer program product for use in conjunction with a computer system, the computer program product comprising a computer readable storage medium and a computer program mechanism embedded therein, the computer program mechanism comprising:

Application No. 10/005,923
Appeal Brief

a program checking tool for analyzing a computer program to produce one or more warnings (FIG. 1: #150; p. 9, lines 13-30; FIG. 4: #118, 402, 404; p. 16, lines 1-9);

at least one warning mapper for mapping at least one of said warnings into at least one annotation modification (FIG. 1: #142 and 144; p. 9, lines 27-30; FIG. 4: #410; p. 16, lines 9-14);

a program updater for updating the computer program in accordance with the annotation modification so that the number of annotations in the computer program changes (FIG. 1: #140; p. 10, lines 1-3; FIG. 4: #412; p. 16, lines 14-17); and

control instructions for repeatedly invoking the program checking tool, warning mapper and program updater until no warnings produced by the program checking tool are suitable for mapping into an annotation modification (FIG. 1: #130 and 150; p. 15, lines 8-17; p. 16, lines 23-31).

Claim 36

A system for annotating a computer program with at least one annotation, the system comprising:

at least one memory (FIG. 1: #106), at least one processor (FIG. 1: #102) and at least one user interface (FIG. 1: #108), all of which are connected to one another by at least one bus (FIG. 1: #104; p. 8, lines 24-31));

wherein said at least one processor is configured to:

annotate the computer program with at least one annotation (FIG. 1: #132; p. 9, lines 24-30; p. 10, lines 22-28);

apply a program checking tool to the computer program to produce one or more warnings (FIG. 1: #150; p. 9, lines 13-30; FIG. 4: #118, 402, 404; p. 16, lines 1-9);

map at least one of said warnings into at least one annotation modification (FIG. 4: #410; p. 16, lines 9-14);

modify the computer program in accordance with the annotation modification so that the number of annotations in the computer program changes (FIG. 4: #412; p. 16, lines 14-17); and

repeat applying the program checking tool, mapping said warnings and modifying the program until no warnings produced by the program checking tool are suitable for

Application No. 10/005,923
Appeal Brief

mapping into an annotation modification (p. 15, lines 8-17; p. 16, lines 23-31).

Claim 50

A method of annotating a computer program, comprising:

- a) applying a program checking tool to the computer program to produce one or more warnings about potential misapplications of primitive operations in the computer program (FIG. 4: #118, 402, 404; p. 16, lines 1-9);
- b) mapping at least one of said warnings into at least one annotation modification (FIG. 4: #410; p. 16, lines 9-14);
- c) inserting into the computer program said at least one annotation modification, thereby producing a modified computer program (FIG. 4: #412; p. 16, lines 14-17);
- d) repeating each of a), b) and c) until no new warnings are produced in a) that are suitable for mapping into an annotation modification (p. 15, lines 8-17; p. 16, lines 23-31); and
- e) providing a user with the modified computer program in which is found at least one annotation (p. 16, lines 26-29).

Claim 51

A method of annotating a computer program, comprising:

- a) inserting a candidate set of annotations into the computer program by employing a heuristic analysis of the computer program (FIG. 5: #502 and 504; p. 19, lines 24-32);
- b) applying a program checking tool to the computer program to produce one or more warnings about inconsistencies between the computer program and one or more of the annotations (FIG. 4: #118, 402, 404; p. 16, lines 1-9; FIG. 5: #506; p. 20, lines 1-3);
- c) mapping at least one of said warnings into at least one annotation modifications (FIG. 4: #410; p. 16, lines 9-14; FIG. 5: #512; p. 20, lines 5-6);
- d) removing from the computer program an annotation, from said set of candidate annotations, that is mentioned by at least one of said warnings, thereby producing a modified computer program (FIG. 4: #412; p. 16, lines 14-17; FIG. 5: #514; p. 20, lines 6-9);

Application No. 10/005,923
Appeal Brief

e) repeating each of b), c) and d) until no new warnings are produced in b) that are suitable for mapping into an annotation modification (p. 15, lines 8-17; p. 16, lines 23-31; p. 20, lines 9-14); and

f) providing a user with the modified computer program in which is found at least one annotation (p. 16, lines 26-29; p. 20, lines 13-14).

Application No. 10/005,923
Appeal Brief

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

I. Claims 1-8, 14-28, 32-42, and 46-51 are rejected under 35 USC § 103(a) as being unpatentable over USPN 5,423,027 (hereafter Jackson) in view of USPN 6,343,376 (hereafter Saxe).

II. Claim 9 is rejected under 35 USC § 103(a) as being unpatentable over Jackson in view of Saxe and further in view of USPN 6,553,362 (hereafter Saxe 362).

III. Claims 10-13, 29-31, and 43-45 are rejected under 35 USC § 103(a) as being unpatentable over Jackson in view of Saxe and further in view of USPN 6,154,876 (hereafter Haley).

Application No. 10/005,923
Appeal Brief

VII. ARGUMENT

The rejection of claims 1 – 51 is improper, and Applicants respectfully requests withdraw of this rejection.

The claims do not stand or fall together. Instead, Applicants present separate arguments for various independent and dependent claims. Each of these arguments is separately argued below and presented with separate headings and sub-heading as required by 37 C.F.R. § 41.37(c)(1)(vii).

I. Claim Rejections: 35 USC § 103

Claims 10-13, 29-31, and 43-45 are rejected under 35 USC § 103(a) as being unpatentable over Jackson in view of Saxe and further in view of USPN 6,154,876 (hereafter Haley). Applicants respectfully traverse.

To establish a prima facie case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art cited must teach or suggest all the claim limitations. See M.P.E.P. § 2143. Applicants assert that the rejection does not satisfy any of the criteria of MPEP § 2143.

As a precursor to the arguments, Applicants provide an overview of Jackson and Saxe.

Overview of Jackson

By way of background, computer programs are often embellished with specifications that specify dependencies among different components of objects in the program (col. 2, lines 3-37). “These specifications typically take the form of lines added into the code by a programmer or by the checker program” (col. 3, lines 38-40).

Jackson teaches a checker program that finds errors in computer programs (see col. 4, lines 12-18). One question with Jackson is: What is the sequence of steps that

Application No. 10/005,923
Appeal Brief

Jackson uses to find and report errors that occur in a program? Jackson expressly answers this question.

Jackson teaches a specific order or sequence of steps for finding reporting errors. First, the programmer or the checker program adds aspect specifications to the computer program (col. 5, lines 63-67). Second, checker program reviews the aspect specifications and program code to locate errors (col. 5, line 67 – col. 6, line 2). Third, the program checker outputs the identified errors to a user (col. 6, lines 2 – 3).¹

Overview of Saxe

In Saxe, source code of a program is input to a static checker that analyzes the source code for errors (6: 1-5). If the static checker finds no errors, then a user is instructed via a report that the source code is accurate (6: 9-14). On the other hand, if errors are discovered, then the user is informed about the errors and provided guidance on how to correct them (6: 16-22).

Saxe focuses on how the errors in the source code are found. Specifically, the source code is converted into formulae (known as verifications), and these formulae are used to generate "an initial graphical representation of the verification condition by generating a tree-like graph structure called an E-graph" (6:30-42). The E-graph is presented to a prover which uses a pattern matching process to find error conditions in the source code (6:53-67).

No Suggestion/Motivation to Modify/Combine References

For at least the following reasons, no suggestion or motivation exists to modify or combine Jackson in view of Saxe.

First, the independent claims recite a series of steps or elements for annotating a computer program (example, claim 1 recites steps a, b, and c). The claims then recite repeating these steps until no warnings are produced. The Examiner admits that Jackson does not teach or suggest this element of "repeating each of steps a, b and c until no

¹ Applicants will show how the independent claims recite a sequence that is quite different than the steps taught or suggested in Jackson.

Application No. 10/005,923
Appeal Brief

warnings are produced" (see FOA at p. 8). The Examiner, however, introduces Saxe to cure this deficiency in Jackson.

Applicants disagree and argue that no motivation or suggestion exists for this combination. Specifically, Applicants respectfully assert that the Examiner is performing hindsight reconstruction to show the claim element of repeating steps until no warnings are produced. In other words, the Office Action is picking and choosing teachings from Saxe and adding them to Jackson to show the claim elements. On this subject, the case law is clear: One cannot use hindsight reconstruction to pick and choose among isolated disclosures in the prior art to deprecate the claimed invention. *In re Fine*, 837 F.2d 1071, 5 U.S.P.Q.2d 1596 (Fed. Cir. 1988).

For at least these reasons, Applicants respectfully argue that a *prima facie* case of obvious has not been established.

Second, Applicant argues that no teaching or suggestion exists to make the combination because the references are directed to different inventions. Granted, both Jackson and Saxe are broadly directed to program checking, but the methods taught for performing such program checking are completely different. Jackson is directed to a program checker that uses a specific sequence for finding and reporting errors in a program.² By contrast, Saxe does not teach or suggest adding specifications to a program. Saxe is directed to converting source code into formulae (i.e., verifications) and then generating "an initial graphical representation of the verification condition by generating a tree-like graph structure called an E-graph" (6:30-42). The E-graph is presented to a prover which uses a pattern matching process to find error conditions in the source code (6:53-67).

The Examiner must provide *objective evidence*, rather than subjective belief and unknown authority, of the requisite motivation or suggestion to combine or modify the cited references. *In re Lee*, 61 U.S.P.Q.2d. 1430 (Fed. Cir. 2002). Obviousness cannot be established by combining the teachings of the prior art to produce the claimed invention absent some teaching or suggestion supporting the combination. *ACS Hospital Systems*,

² First, the programmer or the checker program adds aspect specifications to the computer program (col. 5, lines 63-67). Second, checker program reviews the aspect specifications and program code to locate errors (col. 5, line 67 - col. 6, line 2). Third, the program checker outputs the identified errors to a user (col. 6, lines 2 - 3)

Application No. 10/005,923
Appeal Brief

Inc. v. Montefiore Hospital, 732 F.2d 1572, 1577, 221 U.S.P.Q. 929, 933 (Fed. Cir. 1984). Such teaching or suggestion does not exist.

For at least these reasons, Applicants respectfully argue that a *prima facie* case of obvious has not been established.

Third, Applicant argues that no teaching or suggestion exists to make the combination because the references are directed to solving completely different problems. In Jackson, the Background section discusses that the prior verification approach for finding errors in programs requires rigorous mathematical techniques, cannot be used for finding a broad range of errors, and does not specify what the program does in formal specification language (1:45 – 2:9). Jackson's invention solves this problem by providing "a debugging tool that is easily implemented and catches a broad scope of errors" (2: 9-12). By contrast, the Background section in Saxe discusses the problems with conventional provers. Such prior provers are memory-intensive and time-intensive. Saxe desires to solve these problems by providing "a prover that is both speedy as well as efficient in its use of memory resource" (1: 44-55).

To establish a *prima facie* case, the Examiner must not only show that the combination includes *all* of the claimed elements, but also a convincing line of reason as to why one of ordinary skill in the art would have found the claimed invention to have been obvious in light of the teachings of the references. *Ex parte Clapp*, 227 U.S.P.Q. 972 (B.P.A.I. 1985). In light of the completely different inventions and problems being solved in Jackson and Saxe, no suggestion or motivation exists to combine or modify these references.

For at least these reasons, Applicants respectfully argue that a *prima facie* case of obvious has not been established.

All Elements Not Taught or Suggested

All of the elements of the claims are not taught or suggested in Jackson in view of Saxe. In other words, evening assuming *arguendo* that Jackson and Saxe are successfully combinable (which they are not), the alleged combination does not teach or suggest all the elements in the claims. Examples for various independent and dependent claim groups are provided below.

Application No. 10/005,923
Appeal Brief

Argument A: Independent Claims 1, 22, 36, 50, 51

Jackson in view of Saxe does not teach or suggest all the elements of independent claims 1, 22, 36, 50, and 51. Applicants select independent claim 1 for discussion.

Applicants argue that the teachings in Jackson are very different than the recitations in claim 1. Applicants reiterate that Jackson teaches three steps for locating and reporting errors: (1) use program checker to add lines of aspect specification to a computer program, (2) use program checker to process the computer program and find errors, and (3) report errors to a user. For numerous reasons, these steps are quite different than the steps in claim 1. First, claim 1 recites (a) applying a program checking tool to the computer program to produce a warning. Notice that step (a) in claim 1 is the third step in Jackson. Then, claim 1 recites (b) mapping the warning to an annotation modification. Jackson never performs this step. In Jackson, once an error is discovered, it is provided to a user. Then, claim 1 recites (see steps c and d) modifying the computer program until no warnings are produced. Jackson never performs this step.

Once the checker program in Jackson discovers an error, the error is reported to a user. Jackson does not teach that the checker program maps a discovered error to an annotation and then modifies the annotation to cure the error. FIG. 11 in Jackson is a flow diagram that teaches what the checker program does with errors once they are discovered. Applicants reproduce portions of Jackson's specification that discuss FIG. 11:

When the aspect view specification and the flow specification are completed, a programmer passes the entire annotated program or only a portion of the program through the checker to check for errors. The checker generates both error messages and warning messages. The error messages indicate faults in the aspect specification text. These may be lexical errors, syntax errors or violations of consistency checks. Warning messages indicate faults in the aspect specification text such as the failure to include all the aspects of results objects and in flow

Application No. 10/005,923
Appeal Brief

specification. Conjecture messages, on the other hand, note discrepancies between the code and the aspect specification. The conjecture messages serve to identify the missing dependency and to indicate procedure wherein the error is found.

The checker locates errors by the method described in FIG. 11 by initially constructing a flow for the code in step 160 and comparing the constructed flow with the specified flow in step 162. The checker reports an error in step 164 if the constructed flow does not correspond to the flow specification. (Emphasis added: Col. 13, lines 21 – 42).

Thus, Jackson expressly teaches that the checker program locates errors in the computer program and then reports these errors to the user. Nowhere does Jackson teach or suggest that the checker program maps the errors to annotations, then modifies the annotations to cure the errors. Specifically, the flow diagram in FIG. 11 of Jackson expressly shows that once errors are reported in block 164, flow ends (i.e., the arrow from block 164 links to the "end" block).

Applicants respectfully ask the Board of Appeals to focus on the sequence of steps that Jackson teaches and then compare this sequence to steps a, b, and c in claim 1. Again, the first step in Jackson is to modify the program by adding lines of aspect specification. Then, the program checker finds errors in the programs and reports errors to the user. By contrast, claim 1 does not recite this sequence of steps. In claim 1, the first step is to apply the program checker to the program to produce warnings. The second step is to map the warnings to an annotation modification. The third step is to modify the program with an annotation modification. Again, Jackson's first step is to modify the program by adding lines of aspect specification. The third step in claim 1 is modify the program with an annotation modification. Jackson does not teach or suggest the steps of a, b, and c as recited in claim 1.

For at least these reasons, independent claims 1, 22, 36, 50, and 51 are allowable over Jackson in view of Saxe. The dependent claims are allowable for at least the reasons given in connection with the independent claims.

Application No. 10/005,923
Appeal Brief

Response to Examiner's Argument

The Examiner argues that Jackson's step of using the program checker to add lines of aspect specification to a computer program is equivalent to the claim recitation of "mapping a warning into an annotation modification" (see FOA at pages 3-4). Applicants respectfully disagree because the express language of Jackson teaches that errors are sent to a user, not mapped to annotations. The Examiner's interpretation is contrary to the language in Jackson itself (see FIG. 11 of Jackson, the summary, the specification at col. 13, lines 21 - 42, etc.).

Further, the Examiner repeatedly cites Jackson at col. 5, lines 63-67. Applicants respectfully encourage the Board to read this citation. This section of Jackson actually supports the position of the Applicants. Namely, Jackson teaches three steps for locating and reporting errors: (1) use program checker to add lines of aspect specification to a computer program, (2) use program checker to process the computer program and find errors, and (3) report errors to a user. Nowhere does this section of Jackson teach or suggest mapping errors to an annotation and then modifying the annotation/program to cure the errors. In Jackson, errors are sent to a user.

Argument B: Independent Claims 1, 22, 36, 50, 51

Jackson in view of Saxe does not teach or suggest all the elements of independent claims 1, 22, 36, 50, and 51. Applicants select independent claim 1 for discussion.

The Office Action admits that "Jackson does not expressly disclose repeating each of steps a, b and c until no warnings produced" (see OA p. 4). Applicants agree with this admission. The Office Action, however, attempts to cure this deficiency with Saxe. Applicants respectfully disagree.

First, Jackson actually teaches away from repeating each of steps a, b, and c until no warnings are produced. The specification in Jackson (example, see FIG. 11 and col. 13, lines 21 - 42) teaches that errors are sent to the user, not cured with the checker program.

Application No. 10/005,923
Appeal Brief

For at least these reasons, independent claims 1, 22, 36, 50, and 51 are allowable over Jackson in view of Saxe. The dependent claims are allowable for at least the reasons given in connection with the independent claims.

Second, the Office Action cites Saxe at col. 7, lines 19-26 for teaching the limitation of d) repeating each of steps a, b, and c. This section of Saxe is reproduced below for convenience:

In an extension of the present invention, detection by the prover 204 of an error or inconsistency in the formula being analyzed permits the output of the prover to be used as the basis for assisted amelioration or elimination of the diagnosed formal error, as shown at 209. The edited source code may optionally be fed back to the verification condition generator 202, causing the process to repeat from the beginning as described earlier.

This section of Saxe merely teaches that a section of source code is fed back into a verification condition generator. By contrast, claim 1 recites repeating each of three steps: (a) applying a checking tool to produce one or more warnings, (b) mapping at least one warning into an annotation modification, and (c) modifying the computer program in accordance with the annotation modification. Nowhere does Saxe teach or suggest that steps a, b, and c of claim 1 are repeated until no warnings are produced. Instead, Saxe merely teaches that a section of source code is fed back into a verification generator.

For at least these reasons, independent claims 1, 22, 36, 50, and 51 are allowable over Jackson in view of Saxe. The dependent claims are allowable for at least the reasons given in connection with the independent claims.

Dependent claims 3 and 24

Dependent claims 3 and 24 recite additional recitations that are not taught in Jackson and Saxe. Claim 3 is selected for discussion.

Claim 3 recites that the computer program is modified by inserting an annotation "that is produced by mapping at least one of said warnings about potential

Application No. 10/005,923
Appeal Brief

misapplications of primitive operations into an annotation modification." In other words, claim 3 recites that a warning is mapped to an annotation modification, and this annotation modification is inserted into the computer program. The Examiner cites FIG. 6 of Jackson and column 3, lines 43-48. Applicants respectfully disagree.

First, column 3, lines 43-48 of Jackson merely teaches that errors are detected of considerable broader scope than those caught by type checking. Nowhere does this section of Jackson teach anything whatsoever about mapping a warning into an annotation modification and then inserting this annotation modification into the computer program.

Second, FIG. 6 in Jackson provides an example procedure that exports the abstract object type "intset." Nowhere does this section of Jackson teach anything whatsoever about mapping a warning into an annotation modification and then inserting this annotation modification into the computer program.

Dependent Claims 6 and 27

Dependent claims 6 and 27 recite additional recitations that are not taught in Jackson and Saxe. Claim 6 is selected for discussion.

Claim 1 recites that a computer program is modified with at least one annotation modification. Claim 6 then recites that computer program is modified by "removing from the computer program one of said heuristically derived annotations identified by said at least one annotation modification." The Examiner cites FIG. 2 of Jackson and column 12, line 61 – column 13, line 2. Applicants respectfully disagree.

FIG. 2 and the cited section of Jackson teach that a user can omit part of the program that is not to be checked. Notice how this teaching in Jackson is quite different than the recitations in claim 6. Again, claim 6 recites that a program (not a user) removes "heuristically derived annotation identified by said at least one annotation modification." By contrast, Jackson teaches that a user identifies a portion of a program not to be checked. Jackson does not teach that a program or even a user removes "heuristically derived annotations identified by said at least one annotation modification."

Application No. 10/005,923
Appeal Brief

II. Claims Rejection: 35 USC § 103

Claim 9 is rejected under 35 USC § 103(a) as being unpatentable over Jackson in view of Saxe and further in view of USPN 6,553,362 (hereafter Saxe 362). Applicants respectfully traverse.

Claim 9 depends from claim 1. As noted above in connection with section I, Jackson in view of Saxe does not teach or suggest all the limitations of claim 1. Saxe '362 fails to cure the deficiencies noted above. Thus, for at least the reasons given in connection with claim 1, claim 9 is allowable.

III. Claims Rejection: 35 USC § 103

Claims 10-13, 29-31, and 43-45 are rejected under 35 USC § 103(a) as being unpatentable over Jackson in view of Saxe and further in view of USPN 6,154,876 (hereafter Haley). Applicants respectfully traverse.

To establish a prima facie case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art cited must teach or suggest all the claim limitations. See M.P.E.P. § 2143. Applicants assert that the rejection does not satisfy any of the criteria of MPEP § 2143.

Claims 10-13 depend from claim 1; claims 29-31 depend from claim 22; and claims 43-45 depend from claim 36. As noted above in connection with section I, Jackson in view of Saxe does not teach or suggest all the limitations of claims 1, 22, and 36. Haley fails to cure the deficiencies noted above. Thus, for at least the reasons given in connection with claims 1, 22, and 36, claims 10-13, 29-31, and 43-45 are allowable.

Application No. 10/005,923
Appeal Brief

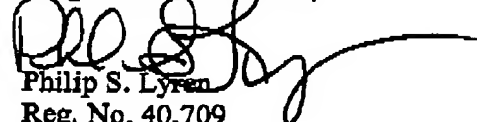
CONCLUSION

In view of the above, Applicants respectfully request the Board of Appeals to reverse the Examiner's rejection of all pending claims.

Any inquiry regarding this Amendment and Response should be directed to Philip S. Lyren at Telephone No. (281) 514-8236, Facsimile No. (281) 514-8332. In addition, all correspondence should continue to be directed to the following address:

Hewlett-Packard Company
Intellectual Property Administration
P.O. Box 272400
Fort Collins, Colorado 80527-2400

Respectfully submitted,


Philip S. Lyren
Reg. No. 40,709
Ph: 281-514-8236

CERTIFICATE UNDER 37 C.F.R. 1.8

The undersigned hereby certifies that this paper or papers, as described herein, is being transmitted to the United States Patent and Trademark Office facsimile number 571-273-8300 on this 3rd day of March, 2006.

By 
Name: Carrie McKerley

Application No. 10/005,923
Appeal Brief

VIII. Claims Appendix

1. (original) A method of annotating a computer program, comprising:
 - a) applying a program checking tool to the computer program to produce one or more warnings;
 - b) mapping at least one of said warnings into at least one annotation modification;
 - c) modifying the computer program in accordance with said at least one annotation modification so that the number of annotations in the computer program changes, thereby producing a modified computer program;
 - d) repeating each of steps a), b) and c) until no warnings produced in step a) are suitable for mapping into an annotation modification; and
 - e) providing a user with the modified computer program in which is found at least one annotation.
2. (original) The method of claim 1 wherein at least a subset of said warnings are warnings about potential misapplications of primitive operations in the computer program.
3. (original) The method of claim 2 wherein, prior to said mapping, said warnings about potential misapplications of primitive operations in the computer program are identified, and said modifying comprises inserting into the computer program at least one annotation that is produced by mapping at least one of said warnings about potential misapplications of primitive operations into an annotation modification.
4. (original) The method of claim 1 wherein, prior to said applying, a candidate set of heuristically derived annotations is inserted into the computer program.
5. (original) The method of claim 4 wherein at least a subset of said warnings are warnings about inconsistencies between the computer program and one or more of the annotations.

Application No. 10/005,923
Appeal Brief

6. (original) The method of claim 5 wherein said warnings about inconsistencies between the computer program and one or more of the annotations are identified, and said modifying comprises removing from the computer program one of said heuristically derived annotations identified by said at least one annotation modification.
7. (original) The method of claim 4 wherein said set of candidate annotations comprises a candidate invariant for a variable f .
8. (original) The method of claim 4 wherein said set of candidate annotations comprises at least one candidate precondition for a procedure in said computer program.
9. (original) The method of claim 4 wherein said set of candidate annotations comprises at least one candidate postcondition for a procedure in said computer program.
10. (original) The method of claim 7 wherein said candidate invariant is of the form $f1 = \text{null}$.
11. (original) The method of claim 7 wherein said candidate invariant comprises an expression that includes a comparison operator.
12. (original) The method of claim 11 wherein said comparison operator is selected from the group consisting of: $<$, $<=$, $=$, $!=$, $>=$ and $>$.
13. (original) The method of claim 11 wherein said expression includes an operand selected from the group consisting of: a variable declared earlier in a same class of the computer program; any one of the constants -1, 0, 1; and a constant dimension in an array allocation expression in the computer program.
14. (original) The method of claim 1 wherein at least one of said warnings includes an explanation.

Application No. 10/005,923
Appeal Brief

15. (original) The method of claim 1 wherein at least one of said annotations in said modified computer program includes an explanation.
16. (original) The method of claim 6 wherein said removing comprises commenting out one of said heuristically derived annotations from the computer program.
17. (original) The method of claim 16 wherein said removing additionally comprises adding an explanatory comment into one of said heuristically derived annotations from the computer program.
18. (original) The method of claim 3 wherein said annotation includes an explanatory comment.
19. (original) The method of claim 1 wherein said program checking tool is a type checker.
20. (original) The method of claim 1 wherein said program checking tool is an extended static checker.
21. (original) The method of claim 1 wherein said program checking tool comprises a verification condition generator and a theorem prover.
22. (original) A computer program product for use in conjunction with a computer system, the computer program product comprising a computer readable storage medium and a computer program mechanism embedded therein, the computer program mechanism comprising:
- a program checking tool for analyzing a computer program to produce one or more warnings;
 - at least one warning mapper for mapping at least one of said warnings into at least one annotation modification;

Application No. 10/005,923
Appeal Brief

a program updater for updating the computer program in accordance with the annotation modification so that the number of annotations in the computer program changes; and

control instructions for repeatedly invoking the program checking tool, warning mapper and program updater until no warnings produced by the program checking tool are suitable for mapping into an annotation modification.

23. (original) The computer program product of claim 22 wherein at least a subset of said warnings are warnings about potential misapplications of primitive operations.

24. (original) The computer program product of claim 23, wherein said at least one warning mapper includes instructions for identifying said warnings about potential misapplications of primitive operations, and the program updater includes instructions for inserting into the computer program an annotation that the warning mapper produces by mapping at least one of said warnings about potential misapplications of primitive operations into an annotation modification.

25. (original) The computer program product of claim 22, further including a heuristic annotation generator for generating and inserting a candidate set of heuristically derived annotations into the computer program.

26. (original) The computer program product of claim 25 wherein at least a subset of said warnings are warnings about inconsistencies between the computer program and one or more of the annotations.

27. (original) The computer program product of claim 25 wherein the warning mapper includes instructions for identifying said warnings about inconsistencies between the computer program and one or more of the annotations, and the program updater includes instructions for removing from the computer program one of said heuristically derived annotations identified by said annotation modification.

Application No. 10/005,923
Appeal Brief

28. (original) The computer program product of claim 25 wherein said candidate set of annotations comprises a candidate invariant for a variable f .
29. (original) The computer program product of claim 28 wherein said candidate invariant comprises an expression that includes a comparison operator.
30. (original) The computer program product of claim 29 wherein said comparison operator is selected from the group consisting of: $<$, $<=$, $=$, $!=$, $>=$ and $>$.
31. (original) The computer program product of claim 29 wherein said expression includes an operand selected from the group consisting of: an earlier declared variable in a same class of the computer program; any one of the constants -1, 0, 1; and a constant dimension in an array allocation expression in the computer program.
32. (original) The computer program product of claim 27 wherein said instructions for removing comprise instructions for commenting out one of said heuristically derived annotations from the computer program.
33. (original) The computer program product of claim 22 wherein said program checking tool is a type checker.
34. (original) The computer program product of claim 22 wherein said program checking tool is an extended static checker.
35. (original) The computer program product of claim 22 wherein said program checking tool comprises a verification condition generator and a theorem prover.
36. (original) A system for annotating a computer program with at least one annotation, the system comprising:
at least one memory, at least one processor and at least one user interface, all of which are connected to one another by at least one bus;

Application No. 10/005,923
Appeal Brief

wherein said at least one processor is configured to:
annotate the computer program with at least one annotation;
apply a program checking tool to the computer program to produce one or more warnings;
map at least one of said warnings into at least one annotation modification;
modify the computer program in accordance with the annotation modification so that the number of annotations in the computer program changes; and
repeat applying the program checking tool, mapping said warnings and modifying the program until no warnings produced by the program checking tool are suitable for mapping into an annotation modification.

37. (original) The system of claim 36 wherein at least a subset of said warnings are warnings about potential misapplications of primitive operations.

38. (original) The system of claim 37, wherein said at least one processor identifies said warnings about potential misapplications of primitive operations, and inserts into the computer program an annotation that a warning mapper produces by mapping at least one of said warnings about potential misapplications of primitive operations into an annotation modification.

39. (original) The system of claim 36, wherein said at least one processor further causes a heuristic annotation generator to generate a candidate set of heuristically derived annotations.

40. (original) The system of claim 38 wherein at least a subset of said warnings are warnings about inconsistencies between the computer program and one or more of the annotations.

41. (original) The system of claim 39 wherein the at least one processor removes from the computer program one of said heuristically derived annotations identified by said annotation modification.

Application No. 10/005,923
Appeal Brief

42. (original) The system of claim 39 wherein said candidate set of annotations comprises a candidate invariant for a variable f .
43. (original) The system of claim 42 wherein said candidate invariant comprises an expression that includes a comparison operator.
44. (original) The system of claim 43 wherein said comparison operator is selected from the group consisting of: $<$, $<=$, $=$, $!=$, $>=$ and $>$.
45. (original) The system of claim 43 wherein said expression includes an operand selected from the group consisting of: an earlier declared variable in a same class of the computer program; any one of the constants -1, 0, 1; and a constant dimension in an array allocation expression in the computer program.
46. (original) The system of claim 41 wherein the at least one processor comments out one of said heuristically derived annotations from the computer program.
47. (original) The system of claim 36 wherein said program checking tool is a type checker.
48. (original) The system of claim 36 wherein said program checking tool is an extended static checker.
49. (original) The system of claim 36 wherein said program checking tool comprises a verification condition generator and a theorem prover.
50. (original) A method of annotating a computer program, comprising:
a) applying a program checking tool to the computer program to produce one or more warnings about potential misapplications of primitive operations in the computer program;

Application No. 10/005,923
Appeal Brief

- b) mapping at least one of said warnings into at least one annotation modification;
- c) inserting into the computer program said at least one annotation modification, thereby producing a modified computer program;
- d) repeating each of a), b) and c) until no new warnings are produced in a) that are suitable for mapping into an annotation modification; and
- e) providing a user with the modified computer program in which is found at least one annotation.

51. (original) A method of annotating a computer program, comprising:

- a) inserting a candidate set of annotations into the computer program by employing a heuristic analysis of the computer program;
- b) applying a program checking tool to the computer program to produce one or more warnings about inconsistencies between the computer program and one or more of the annotations;
- c) mapping at least one of said warnings into at least one annotation modifications;
- d) removing from the computer program an annotation, from said set of candidate annotations, that is mentioned by at least one of said warnings, thereby producing a modified computer program;
- e) repeating each of b), c) and d) until no new warnings are produced in b) that are suitable for mapping into an annotation modification; and
- f) providing a user with the modified computer program in which is found at least one annotation.

Application No. 10/005,923
Appeal Brief

IX. EVIDENCE APPENDIX

None.

Application No. 10/005,923
Appeal Brief

X. RELATED PROCEEDINGS APPENDIX

None.